I/O System and Files

Vittayasak Rujivorakul

Lecture 10

I/O System and Files

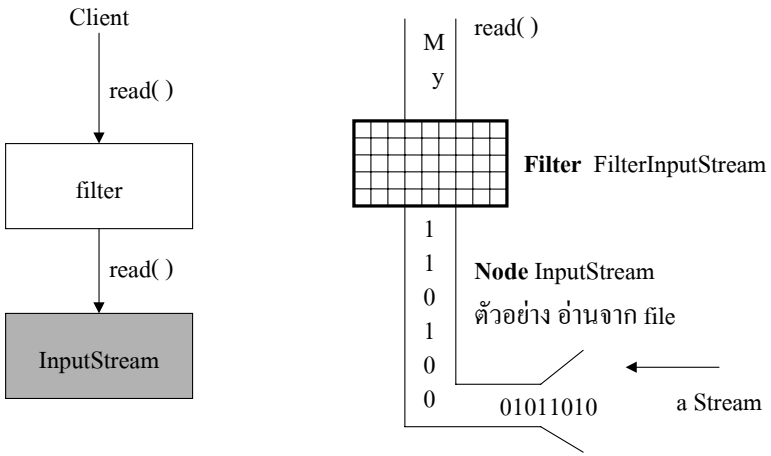
Vittayasak Rujivorakul

Lecture 10

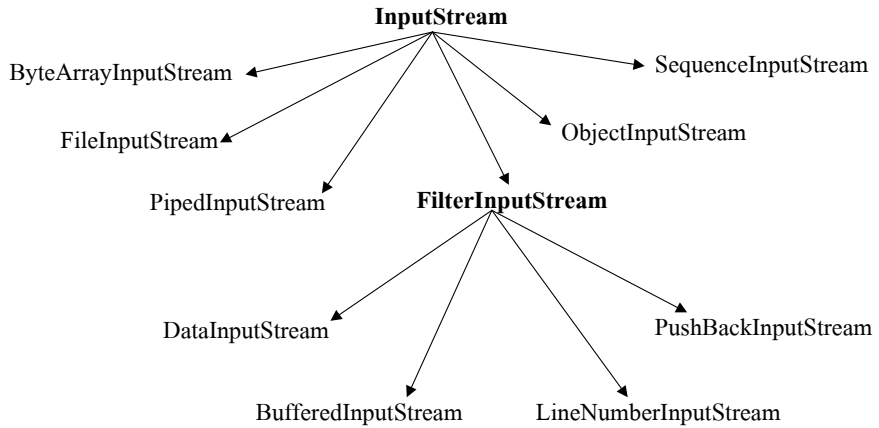
Input and Output

- Java แบ่ง classes ที่ใช้กับ I/O ออกเป็น
  - Stream ของ input กับ output ที่จัดการกับข้อมูล 8 bit binary (source ของ หรือ destination สำหรับ byte data)
    - **ทุก class** ที่สืบทอดจาก InputStream จะ read ( ) method ที่จะอ่าน bytes หรือ **array** ของ bytes
    - **ทุก class** ที่สืบทอดจาก OutputStream จะมี write ( ) method ที่จะเขียน bytes หรือ **array** ของ bytes
  - Reader/Writer classes ที่จัดการกับ String และ 16-bit Unicode character
- Java ยังเตรียม class อื่น ที่เพิ่มความสะดวกในการทำงานกับ I/O ซึ่งต้องใช้งานร่วมกับ I/O classes ข้างต้น

พื้นฐานการทำงานของ Stream



InputStream Class Hierarchy



InputStream

- **InputStream เป็น abstract class** ที่ประกอบด้วย methods ต่อไปนี้
  - int read ( ) throws IOException
    - **ทำการอ่าน 1 byte จาก input**
  - int read (byte [ ] buffer) throws IOException
    - **ทำการอ่าน array ของ bytes จาก input**
  - long skip (long n) throws IOException
    - **ทำการกระโดดข้ามไป n bytes**
  - int available ( ) throws IOException
    - **หาขนาดที่ยังเหลือให้อ่าน**
  - void close ( ) throws IOException
    - **ทำการปิด stream**

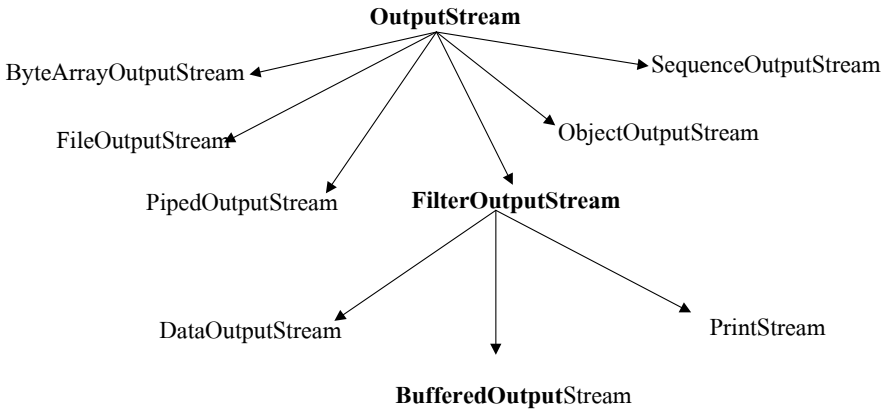
InputStream Categories

- อาจแบ่ง InputStream ได้ออกเป็น 2 กลุ่ม
- **Physical Input Streams :** อ่านข้อมูลจากแหล่งของ input จริง (เช่น buffer, file, และ pipe)
  - ByteArrayInputStream (Byte [ ] buffer)
  - ByteArrayInputStream (Byte [ ] buffer, int offset, int count)
  - FileInputStream (File f)
  - FileInputStream (String filename)
  - PipedInputStream(PipedOutputStream p)

InputStream Categories (ต่อ)

- **Virtual Input Streams :** ต้องทำงานเชื่อมโยงกับ physical input stream เพื่อช่วย เพิ่มขยายหน้าที่ และความสะดวกในการใช้งาน
  - SequenceInputStream **ทำการต่อ 2 input** streams เข้าด้วยกัน
  - ObjectInputStream **ทำการอ่าน** serialized object
  - FilterInputStream และ subclasses ของมัน
    - PushbackInputStream **ยอมให้มีการคืน input ที่อ่าน**แล้วกลับลงสู่ buffer
    - BufferedInputStream ทำการอ่านจาก input stream ในลักษณะ block เพื่อ**เพิ่มประสิทธิภาพ**ในการทำงาน
    - DataInputStream ช่วยในการอ่าน primitive data type **แปลงข้อมู, sbit ให้เป็น primitive ตามที่ต้องการ**

OutputStream Class Hierarchy



OutputStream

- **OutputStream เป็น abstract class** ที่ประกอบด้วย methods ต่อไปนี้
  - void write (int b) throws IOException
    - **ทำการเขียน 1 byte**
  - void write (byte [ ] buffer) throws IOException
    - **ทำการเขียน array ของ bytes**
  - void flush ( ) throws IOException
    - **ทำการปัด output ทั้งหมดออกสู่ buffer**
  - void close ( ) throws IOException
    - **ทำการปิด stream**

Type of OutputStream

- ในทิศทางกลับกันกับ **InputStream มี OutputStream ที่สอดคล้อง**
  - ByteArrayOutputStream ( ) / / buffer default size = 32 bytes
  - ByteArrayOutputStream (int size) / / buffer ตามขนาดที่กำหนด
  - FileOutputStream (File f)
  - FileOutputStream (String filename)
  - PipedOutputStream (PipedInputStream p)

File Object

- File class เป็น class ที่จัดการกับ file โดยที่จะมี method ต่างๆ ช่วยให้ข้อมูลเกี่ยวกับ file นั้น
- **สร้าง File Object (จาก java.io package) โดย**
  - File infile = new File (“sample.data’) ;
    - โดยใช้ชื่อของ file ที่ต้องการอ่านเป็น argument ของ constructor
  - File infile = new File

    (“C : \ \ Lecture \ \ java \ \ Starbucks \ \ “, “sample.data”) ;
    - ระบุ directory ของ file ใน Window Platform
    - ใน Unix ใช้ / Lecture / java/Satrbucks/
  - **หากอ้างถึง file ที่ไม่มีอยู่ File Object จะถูก set เป็น null**

File Object

- **File Object สามารถใช้อ้างถึง directory ได้ด้วย**

```
File directory = new File (“. . \ \ test”) ;
String filename [ ] = directory.list ( ) ;
for (int j = 0 ; j < filenam.length ; j + +)
    System.out.println (filename [ j ] ) ;
```

- **เราสามารถตรวจสอบได้ว่า File Object นั้นอ้างถึง File หรือ Directory โดยเรียก isFile ( ) method**

```
File file = new File (“. . \ \ test”) ;
if (file.isFile ( ) )
    System.out.println (“[ am a file “) ;
else
    System.out.println (“[am a directory”) ;
```

File Utilities

- **File names :**
  - getName ( ), getPath ( ), getAbsolutePath ( ), getPaent ( ), renameTo (File)
- **File Tests (return boolean)**
  - exists ( ), canWrite ( ), canRead ( ), isFile ( ), isDirectory ( ), isAbsolute ( )
- **File Information**
  - lastModified ( ), length ( ), delete ( )
- **Directory**
  - mkdir ( ), list ( )

Directory listing

```
import java.io. * ;
public class DirList   {
    public static void main (String args [ ])    {
        try   {
                File path = new File (“ . “) ;
                String [ ] list ;
                if (args.length =  = 0)
                        list = path.list ( ) ;
                else
                        list = path.list (new DirFilter (args [0] ) ) ;
                for (int   i = 0 ;  i  <  list.length ;   i + +   {
                        System.out.println (list [i] ) ;
                }
        }   catch  (Exception e)    {
                e.printStackTrace ( ) ;
        }
    }
}
```


DirFilter

```
Class DirFilter implements FilenameFilter   {
    String filter ;
    DirFilter (String str)   { filter  =  str ;   }
    public boolean accept (File dir,  String name )    {
        String  f  =  new File (name). getName ( ) ;
        return f.indexOf (filter)  1 = -1 ;
    }
}
```


FileDialog Object

- **ใช้** FileDialog Object ใน java.awt package เพื่อให้ผู้ใช้เลือกชื่อ file **หรือ** directory
- **Object** นี้มีอยู่ 2 modes :
  - **LOAD : เพื่อทำการอ่าน data จาก file ที่ถูกเลือก และ**

```
FileDialog fileBox = new FileDialog/mainWindow ;
    “Open”, FileDialog.LOAD) ;
```

  - **SAVE : เพื่อทำการเขียน data ลงใน file ที่ถูกเลือก**

```
FileDialog fileBox = new FileDialog (mainWindow,“Save As”, FileDialog. SAVE) ;
```

การทำงานกับ physical I/O streams

- เพื่อที่จะอ่านหรือเขียน file ได้เราจะต้องสร้าง java stream objects ชนิดใดชนิดหนึ่ง และเชื่อมโยงมันกับ source (I.e., file)
  - Stream = source ของหรือ destination สำหรับ byte data = sequence ของ data items โดยปกติมีขนาด 8-bit bytes
- **ทำการอ่านจาก input stream** (source ของ data) หรือเขียนลงสู่ output stream (destination ของ data)
  - ดังนั้นการอ่าน file จะเชื่อมโยงกับ input stream และการเขียน file เชื่อมโยงกับ output stream

ตัวอย่าง FileOutputStream

- Output เฉพาะข้อมูลชนิด bytes
- ตัวอย่าง (TestFileOutputStream)

```
File outFile  =  new File (“sample.dat”) ;
FileOutputStream outStream  =  new FileOutputStream
      (outFile) ;
byte [ ] byteArray  =  {10, 20, 30, 40, 50, 60, 70, 80} ;
outStream.write (byteArray) ;
outStream.close ( ) ;
```

- หากไม่ปิด file ข้อมูลอาจมีการสูญหายเนื่องจาก data caching

TestFileOutputStream.java

```
import  java.io.*;
class TestFileOutputStream {
   public static void main(String args[]) throws IOException {
            // create a File object
            File outFile = new File(“sample.data”);
            // associate outFile to a new FileOutputStream object
            FileOutputStream outStream = new FileOutputStream(outFile);
            // prepare output data
            byte[]  byteArray = {10, 20, 30, 40, 50, 60, 70, 80};
            // write the whole byte array at one to the file
            outStream.write(byteArray);
            // output the first and 4th byte
            outStream.write(byteArray[0]);
            outStream.write(byteArray[3]);
            // class the stream
            outStream.close();
   }
}
```


FileInputStream

- ในการอ่าน data เข้ามาใน program ทำขั้นตอนกลับกันกับการเขียน
- ตัวอย่าง (TestFileOutputStream)

```
File inFile = new File (“sample.dat”) ;
FileInputStream inStream = new FileInputStream(inFile) ;
int filesize = (int) infile.length ( ) ;
byte [ ] byteArray = new byt [filesize] ;
inStream.read (byteArray) ;
. . .
inStream.close ( ) ;
```

Data Conversion for write

- **เราสามารถทำ data conversion เพื่อที่จะทำการอ่านหรือเขียนข้อมูลลง file ได้เช่น**

```
File outFile  =  new File (“sample.dat”) ;
FileOutputStream outStream = new FileOutputStream(outFile) ;
byte [ ] byteArray =   { (byte)   ‘J’ ,
                         (byte)   ‘a’ ,
                         (byte)   ‘v’ ,
                         (byte)   ‘a’   } ;
outStream.write (byteArray) ;
outStream.close ( ) ;
```

Type cast characters to bytes

Data Conversion for read

- **เมื่ออ่านกลับ หากต้องการแสดงผลเป็น character เราหา conversion type cast จาก byte เป็น char ไม่เช่นนั้นจะให้ค่าเป็นตัวเลข**

```
File  inFile  =  new File (“sample.dat”) ;
FileInputStream inStream  =  new FileInputStream(inFile) ;
int filesize  =  inFile.length ( ) ;
byte [ ] byteArray  =  new byte [filesize] ;
inStream.read (byteArray) ;
for  (int   j = 0 ;  j  < filesize ;   j + +)
        System.out.println ( ( char) byteArray [j]) ;
inStream.close ( ) ;
```

Type cast byte to char เพื่อแสดงผล

การทำงานกับ Virtual I/O Streams

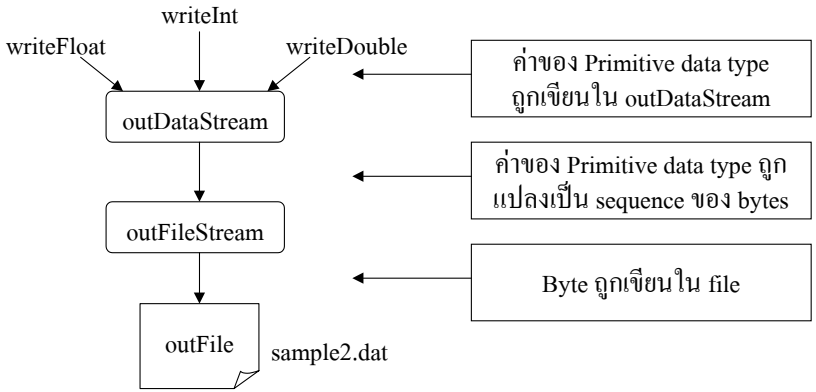
- สามารถที่จะทำการ convert data type อื่น ๆ เป็น byte ได้ (e.g. int = 4 byte) หากแต่ขั้นตอนการทำงานยุ่งยาก
- java เตรียม DataOutputStream object ไว้เพื่อใช้ในการกับ primitive data type
- ตัวอย่าง (TestDataOutputStream)

TestDataOutputStream

```
import  java.io.*;
class TestFileOutputStream {
   public static void main(String args[]) throws IOException {
            File  outFile  =  new File(“sample.dat”);
            FileOutputStream outFileStream = new FileOutputStream(outFile);
            DataOutputStream outDataStream = new DataOutputStream(outFileStream);
            outDataStream.writeInt(987);
            outDataStream.writeLong(1234L);
            outDataStream.writeFloat(1.1f);
            outDataStream.writeDouble(234.8);
            outDataStream.writeChar(‘A’);
            outDataStream.writeBoolean(true);
            outDataStream.close();
   }
}
```

ความสัมพันธ์ของ outFile, outFileStream, outDataStream

```
File                outFile = new File("sample2.dat");
FileOutputStream    outFileStream = new FileOutputStream(outFile);
DataOutputStream    outDataStream = new DataOutputStream(outFileStream);
```

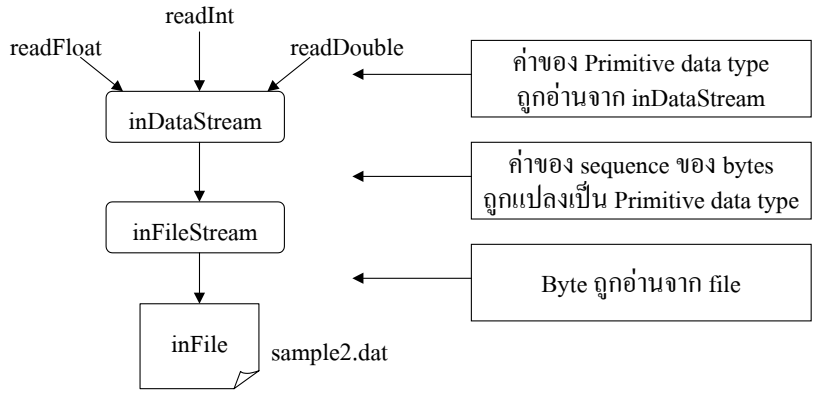


TestDataInputStream

```
import java.io. * ;
class TestDataInputStream   {
    public static void main (String args [ ] )  throws IOExcepttion   {
            File inFile  =  new  File ("sample2.dat") ;
            FileInputStream inFileStream  =  new FileInputStream
    (outFile) ;
            DataInputStream inDataStream = new DataInputStream
    (outFileStream) ;
            System.out.println (inDataStream.readInt ( ) ) ;
            System.out.println (inDataStream.readLong ( ) ) ;
            System.out.println (inDataStream.readFloat ( ) ) ;
            System.out.println (inDataStream.readDouble ( ) ) ;
            System.out.println (inDataStream.readChar ( ) ) ;
            System.out.println (inDataStream.readBoolean ( ) ) ;
            inDataStream.close ( ) ;
    }
}
```


ความสัมพันธ์ของ inFile, inFileStream, inDataStream

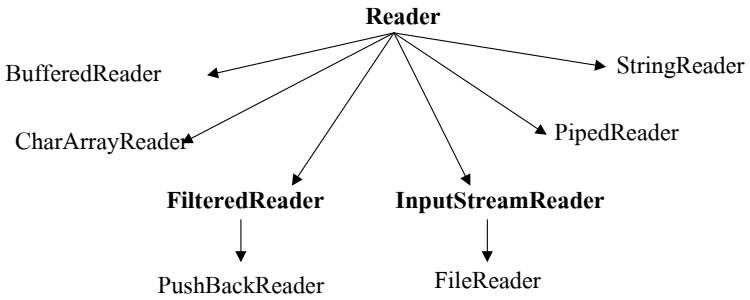
```
File                inFile = new File("sample2.dat");
FileInputStream     inFileStream = new FileInputStream(inFile);
DataInputStream     inDataStream = new DataInputStream(inFileStream);
```



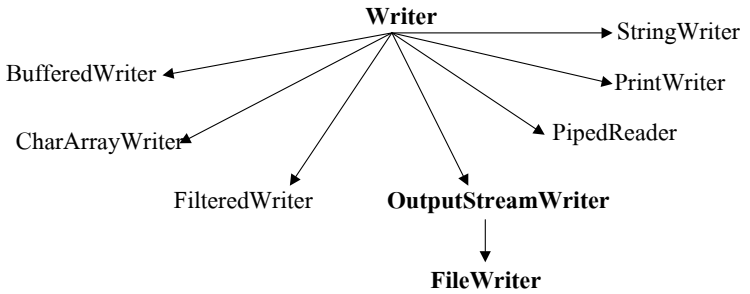
Reader / Writer

- ใน Java ตัวอักษรและสายของอักษรเก็บอยู่ในรูปของ Unicode (16 bits) character โดยที่หากเป็นภาษาอังกฤษใช้มาตรฐาน ISO-8859-1
- Reader / Writer เป็นกลุ่มของ class ที่ทำงานกับ Unicode
- สังเกตการแบ่งย่อย class ต่าง ๆ มีลักษณะใกล้เคียงกับ input และ output streams

Reader Class Hierarchy



Writer Class Hierarchy



Binary VS Text File

- ทั้ง FileOutputStream และ DataOutputStream objects สร้าง binary file นั่นคือ เนื้อข้อมูลถูกเก็บในรูป binary
- สามารถทำการเก็บข้อมูลในรูป ASCII โดยการแปลงให้เป็น string data แล้วจึงเก็บ
- file ที่เก็บข้อมูลในรูปของ ASCII เรียกว่า textfile
- Textfile สามารถอ่านหรือเขียนโดยใช้ text editor เช่น notepad ได้
- Object ที่ใช้ในการสร้าง textfile คือ PrintWriter

PrintWriter

- PrintWriter supports เพียง 2 output methods : print และ println
- สามารถใช้ primitive data type ทุกชนิดเป็น argument ของ method
- โดยที่มันจะทำการแปลงค่าเหล่านั้นเป็น string ก่อนที่จะทำการเขียน
- ในการสร้าง PrintWriter object เช่นเดียวกับ DataOutputStream ต้องการ FileOutputStream เป็น argument

TestPrintStream

```
import java.io. * ;
class TestPrintStream   {
    public static void main (String args [ ] ) throws IOException   {
        File outFile  =  new File (“sample3.dat “ ) ;
        FileOutputStream outFStream  =  new  FileOutputStream outFile) ;
        PrintWriter outStream  =  new PrintWriter (outFStream) ;
        outStream.println (987) ;
        outStream.println (1234L) ;
        outStream.println (1.1f) ;
        outStream.println (231.8) ;
        outStream.println (‘A’) ;
        outStream.println (true) ;
        outStream.close ( ) ;
    }
}
```


Read Text File

- ในการอ่าน textfile เราใช้ FileReader และ BufferedReader objects
- ความสัมพันธ์ระหว่าง FileReader กับ BufferedReader เหมือนกับความสัมพันธ์ของ FileInputStream กับ DataInputStream
- **การ set up**

```
File                inFile   =  new file (“sample3.dat”) ;
FileReader          fReader  =  new  FileReader (infile) ;
BufferedReader bufReader  =  new  BufferedReader (fReader) ;
```

- การอ่าน

```
String str  =  bufReader.readLine ( ) ;
```


TestBufferedReader

```
import java.io. * ;
class TestBufferedReader   {
    public static void main (String args [ ] ) throws IOException   {
        File inFile  =  new File  (“sample3.dat”) ;
        FileReader fReader  =  new FileReader (infile) ;
        BufferedReader bufReader  =  new BufferedReader (fReader) ;
        String str ;
        str = bufReader.readLine ( ) ;
        while (str ! = null)   {
                System.out.println (str) ;
                str = bufReader.readLine ( ) ;
        }
        bufReader.close ( ) ;
    }
}
```


RandomAccessFile

- การทำงานกับ file ในลักษณะ random access
- สร้าง Random Access File ได้จาก
  - โดยใช้ชื่อของ file เป็น argument ของ constructor

    RandomAccessFile (String filename, String mode)
  - โดยใช้ File object เป็น argument ของ constructor

    RandomAccessFile (File filename, String mode)
  - mode ที่เป็นไปได้คือ
    - “ r ” สำหรับอ่านอย่างเดียว หรือ
    - “ rw ” สำหรับทั้งอ่านและเขียน

Methods ที่ใช้ในการ access Random Access File

- long **getFilePointer ( )**
  - คืนตำแหน่งปัจจุบันของ file pointer
- void **seek (long pos)**
  - set file pointer ให้ชี้ที่ตำแหน่ง byte-offset ที่ pos (ต้น file pos = 0)
- long **length ( )**
  - คืนค่าขนาดของ file

การอ่านจาก standard input

```
import java.io. * ;
public class Echo  {
   public static void main (String args [ ] )  {
        Echo  e  =  new Echo ( ) ;
        System.out.println (e,readLine (“Enter something : “ ) ) ;
   }
   public String readLine (String prompt)  {
        BufferedReader  in  =  new  BufferedReader  (
                              new InputStreamReader (System.in ) ) ;
        System.out.print (prompt)  ;
        try  {
                return  in.readLine ( ) ;
        }  catch  (IOException  e)  {  return  null;  } ;
   }
}
```


StreamTokenizer & StringTokenizer

- StreamTokenizer ไม่ใช่ InputStream แต่มีความสามารถในการแตก text file ออกเป็น สายของ tokens โดยมันสามารถที่จะแยก words, ตัวเลขและตัวอักขระพิเศษ
  - ตัวอย่าง Token Test.java
- StringTokenizer ทำหน้าที่คล้ายกันกับ StreamTodenizer แต่ทำกับ String

```
StringTokenizer st  =  new  StringTokenizer (“hello world”) ;
while  (st.hasMoreTokens ( ) )  {
    System.out.println (st.nextToken ( ) ) ;
}
```


Redirecting standard I/O

- สามารถที่จะทำการ redirect standard input, output และ error ได้โดยเรียก static methods ใน class System
  - setln(InputStream)
  - setOut(OutputStream)
  - setErr(PrintStream)
- ตัวอย่าง Redirecting.java

Object Serialization

- Java ได้กำหนด interface Serializable เพื่อช่วยในการเก็บ object ในรูปของ stream
  - Serializable interface เป็น interface class ที่ไม่มี method ใด ๆ แต่ทำหน้าที่เหมือน marker ที่บอกให้รู้ว่า object นั้นสามารถถูกเก็บได้
  - Object จาก class ใดที่ไม่ได้ implements Serialzable ไม่สามารถเก็บได้
- การเก็บ object ให้ถาวรไว้ใน storage เรียกว่า persistence
- การเก็บนี้จะ เก็บสถานะ หรือ data ของ object ที่ serialized เท่านั้น
- ข้อมูลใดที่ไม่ต้องการทำ serialized ต้องกำหนดให้เป็น transient

"transient" keyword

- Object ของ Class ที่ implements Serializable จะถูกเก็บและคืนได้โดยอัตโนมัติไม่ว่าจะประกาศไว้เป็น private, protected หรือ public
- **transient** keyword ช่วยให้เราสามารถเลือกไม่เก็บข้อมูลหรือ subobject ที่เราไม่ต้องการให้มีการเก็บ (เช่น ข้อมูลที่เป็นความลับ, private data) ต้องทำการ mark reference ของ data นั้นเป็น transient

```
public class Myclass implements Serializable   {
        private String custID ;
        private String name ;
        private transient lone total ;
}
```


Object I/O

- ใช้ ObjectOutputStream object เพื่อเขียน object
- ใช้ ObjectInputStream object เพื่ออ่าน object
- Class ที่ต้องการทำให้ persistence ต้อง implements Serializable

```
import java.io. * ;
class Person implements Serializable   {
    / / the rest is the same
}
```

- หาก class นั้นเป็น **subclass** ก็ต้อง implement Serializable

```
class Students extends Person implements Serializable
```


การเขียน Object ลง File

```
/ /  create ObjectOutputStream
File outFile  =  new File ("objects.ser") ;
FileOutputStream outFileStream  =  new FileOutputStream(outFile) ;
ObjectOutputStream outObjectStream  =  new ObjectOutputStream
    (outFileStream) ;
/ / save Person object
Person person = new Person ("Mr. Espresso", 20, 'M') ;
outObjectStream.writeObject (person) ;
/ / save primitive data type into the file
outObjectStream.writeInt (15) ;
```

การอ่าน Object จาก File

```
/ /  create ObjectInputStream
File inFile  =  new File (“objects.ser”) ;
FileInputStream inFileStream  =  new  FileInputStream(inFile) ;
ObjectInputStream inObjectStream  =  new ObjectInputStream (inFileStream) ;
/ / read Person object (must do casting)
Person person  =  (Person) inObjectStream.readObject ( ) ;

/ / read primitive data type into the file
int  j  =  inObjectStream.readInt ( ) ;
```


ClassNotFoundException

- นอกจาก IOException แล้ว readObject method ยัง throws ClassNotFoundException ด้วย

```
public void myMethod ( ) throws IOException,
    ClassNotFoundException  {
```

- สามารถเลือกที่จะ throws ต่อ หรือ catch ได้